Computational geometry and discrete computations

Olivier Devillers

Computational Geometry and Discrete Computations*

Olivier Devillers

INRIA, BP 93, 06902 Sophia-Antipolis cedex
Olivier.Devillers@sophia.inria.fr

Abstract. In this talk we describe some problems arising in practical implementation of algorithms from computational geometry. Going to robust algorithms needs to solve issues such as rounding errors and degeneracies. Most of the problems are closely related to the incompatibility between on one side algorithms designed for continuous data and on the other side the discrete nature of the data and the computations in an actual computer.

1 Introduction.

Computational geometry began in the last seventies, and since has been developed as a whole domain inside theoretical computer science. Research in that area has been very active. Several books [2, 8, 15, 18] and journals are devoted to the field *(IJCGA. CGTA, DCG)*. As suggested by the name itself, "computational geometry" deals with the computation of geometric objects, and more precisely is devoted to the design of geometric algorithms and the study of their complexities.

Below are detailed the main hypotheses which are usually made in computational geometry papers. These hypotheses are very important, they induce simplifications and an abstract framework for the design of algorithms which contribute to the success of the field and its productivity. But these hypotheses are also a major drawback, because they turn the algorithms too abstract and make them far from the reality of programming.

As a main example, I will use the problem of sorting n real numbers. Even if sorting is not really a geometric problem (people can argue that one dimensional geometry is not geometry), it is a good example to illustrate the major points on a very simple and easy to understand problem. In some cases, I will also add some more geometric example.

An important point about geometric algorithms is that they combine geometric and combinatorial aspects. The result of the algorithm is often

* Work supported in part by ESPRIT LTR 21957 (CGAL)

combinatorial, in the sorting example the result is a permutation of size n. But the algorithm involves geometric tests, in our example comparisons of real numbers. In the abstract framework, such a geometric test takes continuous data (e.g.: two real numbers x and y) and returns some discrete data (e.g.: x is smaller/equal/greater than y); such a test or *predicate* often consists in evaluating the sign of some expression (e.g.: sign of $x - y$).

Complexity. Complexity usually means asymptotic complexity, that is the order of magnitude of the number of basic operations needed to solve the problem in terms of the size of the input. For example sorting n numbers costs $O(n \log n)$ comparisons. In such a notation, the big O hides a constant which can be important in practice, but is often forgotten in theoretical papers and never precisely studied.

Lower bounds are usually proved either by evaluating the size of the output, or in the Information Theory Bound model. In the ITB model, we assume that the algorithm is only allowed to make binary decisions and then the smallest number of decisions needed to solve the problem is evaluated using Ben'Or theorems, basically the logarithm of the number of combinatorially different possible results is a lower bound. Such a lower bound only involves binary decision and does not count the amount of work needed between two decisions. For example sorting needs $\Omega(n \log n) = \Omega(\log n!)$ binary decisions.

Model of computation. To count the basic operations, we have to define precisely what a basic operation is. The usual hypothesis is that a geometric predicate can be evaluated in constant time and thus the complexity analysis usually only counts the number of such predicates needed by the algorithm to establish an upper bound on the complexity.

Algorithms are usually designed using a real-number arithmetic, and thus a constant time operation is often an arithmetic operation such as comparison, addition, or multiplication of two real-numbers. This assumption cannot correspond to the reality of a computer because either the bit length of the number is fixed, and then the arithmetic available is not a real-number arithmetic but a *rounded* floating point arithmetic, or the bit length is not fixed, but in that case the time needed to perform an operation is no longer constant.

Sometimes, people even use stronger "basic operations" such as the solving of an system of equations of bounded size and bounded degree (still assumed to need a "constant time").

Lower bounds are also concerned by the model of computation, the Information Theory Bound is valid on Turing machine but not on RAM machine. For example, *bucket sort* works in linear time.

Degenerate cases. Together with the real-number arithmetic, an usual unrealistic hypothesis is the absence of degenerate cases. A situation is said to be degenerated if a small perturbation of the input may not preserve that situation. In the sorting example, the situation is said degenerated if two values are equal. A more geometric example: the fact that three points in the plane are cocircular is not degenerated since three points always define a circle but four cocircular points is a degenerated situation because a small perturbation of one point can move it inside or outside the circle defined by the three others.

When a geometric predicate reduce to the sign of an expression, the positive and negative cases are the two regular answers, and the null case is the degenerated one.

Computational geometry papers usually assume that there is no degenerate cases. This assumption is justified by the fact that in a continuous world such cases have a null probability to occur. But in the computer world where data are discretized, this probability is no longer null, and such cases actually occur.

Organization. I will first describe in Section 2 some problems due to the above hypotheses and the intrinsic discrete nature of geometric data. Then we investigate the two main approaches used to cope with this. In Section 3, we look for means to use discrete data as if we were in a continuous world and, in Section 4, we expose solutions having a more discrete spirit.

2 Real-numbers arithmetic yields non implementable algorithms.

Most of the geometric algorithms rely on geometric theorems which are true in the usual mathematical setting, where the geometric objects have real coordinates. Unfortunately, these theorems are no longer true when real coordinates are replaced by usual machine representable numbers such as `float` or `double`; and the geometric properties are replaced by numerical predicates evaluated using the rounded floating point arithmetic.

For example a simple fact such as *"If points a, b, c are collinear and a, b, d are collinear and $a \neq b$ then a, c, d are collinear"* is no longer reliable,

when collinear means that some arithmetic expression is evaluated to 0 in a rounded computation. For the sorting example, we will assume that the machine comparisons is not transitive (which is hopefully false in actual computers); for the sake of illustration, we assume that entries are decimal numbers, and that our computer is able to compare only integers and thus rounds the number before comparing them.

Thus, most of the geometric theorems are not valid in the computer context, but still, most of the geometric algorithms rely on these geometric theorems. Therefore some situations may appear as incoherent if the input data make some theorem fail. Since such a situation is supposed to be impossible, it is not planned in the program and it fails or falls in an infinite loop or maybe just computes a wrong result.

Many examples of this kind exist in geometric softwares [17]. The above simple example is very representative of the problems, since a common case is that some transitivity property, which is true in theory, is not verified, and the cycle created will correspond to an infinite loop in the program behavior. Imagine that the rounded computation answers that $x > y$, $y > x$, $x < z$ and $y < z$ and bubble sort runs on an array starting by x, y and z. At each new traversal of the array x and y will be swapped, thus if the program runs until the order is fine, it will loop infinitely.

In the following sections we will sketch two solutions to the fact that geometric theorems are not true in the computer model of computation. In Section 3 the model of computation is modified so that relevant theorems still hold. In Section 4 weaker theorems are proved in the computer model of computation and other algorithms based on these new theorems are used.

3 Implementing real-numbers arithmetic.

The most common way to ensure that geometric theorems hold is to compute with enough precision so that the rounding done by the floating point computation cannot introduce a wrong decision (a wrong answer to a predicate).

3.1 Different ways of having precise computations.

Integer and rational arithmetic. *Precise computations* generally means *exact computations*. If the input data are integer (resp. rational) and the predicate corresponds to the evaluation of a polynomial (resp. rational) formula, then computations can be done using exact computations with integer (resp. rational) of arbitrary length.

These exact tests can be achieved using standard libraries of large numbers, or some more specifically geometric methods devoted to some predicates [5, 1].

Arbitrary length floating point numbers. Exact computation can also be achieved with floating point numbers, but of arbitrary length. Such a number of "indefinite precision" is often represented as the sum of a set of usual machine floating point numbers of "fixed precision", and the precise definition of the rounding operation in the IEEE standard is used to make arithmetic operations exactly, using a few machine operations [19, 20].

Algebraic system. Sometimes, the predicate does not reduce to the evaluation of an expression, it can consist in the sign of the solution of some algebraic system, and even an exact arithmetic does not allow to compute that solution exactly. In that case, numerical methods can be used in a robust way. The result can be computed up to some small error, so that the sign is guaranteed [4].

3.2 Filters.

The above methods to reach exact computations solve many of our problems, but they are expensive. A way to reduce the cost of exact computation is to use rounded computation whenever it is possible and to use exact computations only were there is some doubt on the accuracy of the rounded result.

What is a filter? A filter is an approximate evaluator for the result associated to a certifier which may guarantee some property about the precision of that result. Such a filter is often a rounded computation of an expression and an upper bound on the error done during this computation; If the error is smaller than the computed value, then the sign of the expression is certified. If the certifier fails then the exact computation has to be done (or a less cheap but more accurate filter must be tried) [12, 16].

For our weak comparisons described in the introduction, an example of filter is

compute `x-y`
if `x-y` ≥ 2 then return `x>y`
else if `x-y` ≤ -2 then return `x<y`
else return `filter failed`

How is a filter efficient? In practice, rounded computations is most often able to give the right answer, so that the aim of the filter is to find the very few cases where the rounded computation is not accurate enough.

If we still are interested in the comparison test, if we assume that the operands are randomly chosen in interval $[0, N]$ and the filter failed on `x<y` if `x-y`< 2, then the probability of failure is

$$\frac{1}{N^2} \int_0^N dx \int_{\max(0,x-2)}^{\min(N,x+2)} dy \leq \frac{4}{N}$$

Thus if N is large the probability of failure of the filter is small.

Under some hypotheses on the data, the probability of failure of the filter can be evaluated and if it is small enough, it justifies the choice of the filter (see [6] for result for collinearity and cosphericity tests). Hypotheses of random distribution on the data may be unvalid in practice, because some data are by nature degenerated. In that case, the expression whose signe we are looking for can be null, but the probability that it is smaller than ε and non null is still small and an adequate filter can be designed.

4 Designing algorithms for discrete-numbers arithmetic.

An alternative to exact computation consists in ensuring some combinatorial property by some combinatorial way of computing (instead of relying on geometric theorems).

To ensure that the geometric theorems needed by the algorithm are not violated when evaluating predicates, a solution consists in not computing any predicates which can be deduced from former evaluations. For example, if the points `a<b` and `b<c` have been evaluated to be true, and at some point the algorithm asks for `a<c?`, the numerical predicate should not be evaluated because previous evaluations and transitivity imply that the answer is "yes". Such an algorithm is called parsimonious by Knuth [13].

This kind of approach can also be used in a weaker way, that is ensuring weaker properties than the real geometric properties. A good example is Sugihara and Iri's algorithm [21] for Delaunay triangulation. This algorithm verifies during the insertion of each new point that the constructed triangulation is a good topological triangulation (a planar graph, so that each face is a triangle and each edge exists only once). If the geometric predicates are exact then the algorithm construct the Delaunay triangulation, otherwise the algorithm guarantees that it does not fail and actually constructs a topological triangulation. So that another algorithm needing

a triangulation as input can use the result without trouble. Unfortunately, this algorithm does not ensure that the constructed triangulation is topologically equivalent to the Delaunay triangulation of some input, since there exist topological triangulations that cannot be realized as a Delaunay one [7].

Similar phenomena arise in other problems, for example an algorithm computing an arrangement of lines may guarantee some properties on line intersections, but not more complicated ones such as Pappus theorem [13]. Thus the computed arrangement may be not realizable by actual lines, such an arrangement is called an arrangement of pseudo-lines. This difference is important since the combinatorial properties are different [11].

5 Conclusion.

Solving robustness problems in computational geometry which rely on the discrete nature of computer computations is currently a big issue in the field and mobilizes many researchers. Many discussions at the last ACM symposium on computational geometry and the first workshop on applied computational geometry were about this topic. The main issues are the design of fast and exact code to answer geometric predicates (a simple geometric question such as "on which side of this plane does this point lie?") and the creation of new algorithms which do not rely on geometric theorems that are impossible to guarantee using computer arithmetic.

Related works concern also new kinds of analysis. The intrinsic complexity of the geometric predicates can be studied [14, 3], or restricted models of computation were only a small number of well defined geometric predicates are allowed can be developed [10, 9].

Acknowledgments. The author would like to thank the committee of the sixth *Discrete Geometry for Computer Imagery* for inviting him to present this work.

References

1. F. Avnaim, J.-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec. Evaluation of a new method to compute signs of determinants. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C16–C17, 1995.
2. J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique.* Ediscience international, Paris, 1995.

3. C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes, March 1996.
4. Christoph Burnikel, Jochen Könnemann, Kurt Mehlhorn, Stefan Näher, Stefan Schirra, and Christian Uhrig. Exact geometric computation in LEDA. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C18–C19, 1995.
5. K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, 1992.
6. O. Devillers and F. Preparata. A probabilistic analysis of the power of arithmetic filters. Research Report to appear, INRIA, BP93, 06902 Sophia-Antipolis, France, 1996.
7. M. B. Dillencourt. Realizability of Delaunay triangulations. *Inform. Process. Lett.*, 33:283–287, 1990.
8. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.
9. Jeff Erickson. New lower bounds for convex hull problems in odd dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 1–9, 1996.
10. Jeff Erickson and Raimund Seidel. Better lower bounds on detecting affine and spherical degeneracies. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, pages 528–536, 1993.
11. Stefan Felsner. On the number of arrangements of pseudolines. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 30–37, 1996.
12. S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.
13. Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, 1992.
14. G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries in implicit Voronoi diagrams. Research Report CS-96-16, Brown University, Providence, RI, 1996.
15. K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1984.
16. K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proc. 13th World Computer Congress IFIP94*, volume 1, pages 223–231, 1994.
17. D. Michelucci. Arithmetic isuues in geometric computations. In *Proc. 2nd Real Numbers and Computer Conf.*, pages 43–69, Marseille, France, 1996.
18. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.
19. D. Priest. Algorithms for arbitrary precision floating point arithmetic. In *Proc. 10th Symp. on coputer arithmetic*, pages 132–143, 1991.
20. Jonathan R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 141–150, 1996.
21. K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 4:179–228, 1994.